Basic STAMP Tutorial

Version 1.1
Design Engineering Challenge 2007
Introduction

The basic stamp will be used to control the magnet and motor. It will interface with the relay board, which takes stamp input and manages the currents required to run the magnet and motor. You have been given skeleton code, which contains the variable declarations and subroutine flow required to complete this task.

Relay Board

The relay board interfaces with the stamp, responds to stamp I/O, and controls the higher currents required by the magnet and the motor. It connects to the stamp using a 20-pin ribbon cable, which also carries power to the stamp. The motor, magnet, and pushbutton can be connected to the relay board without worrying about polarity.

STAMP Commands

Constants

A constant is a number that doesn’t change throughout the program. Constants make your code more readable when using output. Here’s an example:

```
LED CON 0
HIGH LED
PAUSE 250
LOW LED
```

Even without knowing about input and output, the general feel of the program can be figured out: it sets the LED to HIGH, waits 250ms, and sets the LED to LOW.

DEBUG

DEBUG will output text or variables to a window on the computer that is being used to program the STAMP. To try it out, type the command:

```
DEBUG “Hello World!”
```

When you run the program, a new window will pop up displaying the text:
If the program freezes when you try to close the Debug Terminal, try hitting the reset button on the STAMP board.

**DEBUG DEC**

DEBUG DEC is very similar to DEBUG. It displays output to a window on the computer connected to the stamp. The difference is that DEBUG DEC displays numbers, and DEBUG displays characters. Here’s an example:

```
DEBUG DEC 65
```

This displays the number 65. If instead you tried DEBUG 65, you would see the letter A.

**Variables**

The Basic STAMP supports three data types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>1 bit</td>
<td>0 or 1</td>
</tr>
<tr>
<td>BYTE</td>
<td>8 bits</td>
<td>0 – 255</td>
</tr>
<tr>
<td>WORD</td>
<td>16 bits (2 bytes)</td>
<td>0 – 65,535</td>
</tr>
</tbody>
</table>
To assign variables, use the command VAR (TYPE). For example, to make a BYTE variable named myCount, use:

```
myCount VAR BYTE
```

You can display variable values using the DEBUG command:

```
myCount VAR BYTE
myCount = 5
DEBUG DEC myCount
```

![Figure 2 - Displaying a Variable](image)

**Labels and GOTO**

Labels identify a point in your code. They are then used by the GOTO and GOSUB commands to direct program execution to that point. Here’s an example:

```
loop:
    DEBUG DEC 5
GOTO loop

end
```

This program will run the command DEBUG DEC 5, then branch back to loop, which is just above the DEBUG command. It will then run DEBUG DEC 5 again, and branch back to loop. It will then run DEBUG
DEC 5 again, and branch back to loop. This situation is called an infinite loop, and in most cases it’s not a very good idea. In fact, it’s not used anywhere in the skeleton code except for the main program loop.

---

**GOSUB - RETURN**

GOSUB is similar to GOTO, except that GOSUB keeps track of where the program came from. The RETURN command can then be used to send the program execution back to that point. Here’s an example program that outputs 01:

```plaintext
myCount VAR BYTE
myCount = 0
GOSUB displayMyCount
myCount = 1
GOSUB displayMyCount
END

displayMyCount:
    DEBUG "myCount value: 
    DEBUG DEC myCount,CR
    return
```

**IF-THEN-GOTO**

This command takes the form:

IF (conditions) THEN (branch)

The conditions are the situations in which the program will branch. There are six important comparisons:

<table>
<thead>
<tr>
<th>Comparison Operator</th>
<th>True When</th>
<th>False When</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = Y</td>
<td>X and Y are equal</td>
<td>X and Y are different</td>
</tr>
<tr>
<td>X &gt; Y</td>
<td>X is greater than Y</td>
<td>X is equal to or less than Y</td>
</tr>
<tr>
<td>X &lt; Y</td>
<td>X is less than Y</td>
<td>X is equal to or greater than Y</td>
</tr>
<tr>
<td>X &lt;&gt; Y</td>
<td>X is not equal to Y</td>
<td>X is equal to Y</td>
</tr>
<tr>
<td>X &lt;= Y</td>
<td>X is less than or equal to Y</td>
<td>X is greater than Y</td>
</tr>
<tr>
<td>X &gt;= Y</td>
<td>X is greater than or equal to Y</td>
<td>X is less than Y</td>
</tr>
</tbody>
</table>

The branch clause is essentially a GOTO statement. It will take your program flow and point it to that label. The difference is that this branch only happens when the conditions are met. We can devise a way for breaking out of an infinite loop:
myCount  VAR  BYTE
myCount = 0

loop:
    DEBUG DEC myCount,CR
    IF myCount = 5 THEN leaveLoop
    myCount = myCount + 1
    GOTO loop

leaveLoop:
    DEBUG "Exited the Loop!"
end

Figure 3 - Looping and IF-THEN in action
Skeleton Code

The skeleton code uses all of the concepts discussed above. It can be divided into three main sections:

1. Initialization
2. Main Program Loop
3. Subroutines

Initialization

The program declares all variables and constants in the initialization stage, and then sets all output pins to a known state. You will be using four output pins: three for the motor and one for the magnet. These output pins are all declared as constants to increase readability of the code. You will also be using one input pin for the pushbutton, which is declared as a variable. The delays used for timing are also declared as constants at the top of the code. This makes it easy to change these delays in the future, when you’ll need to tweak the crane’s timing.

The next stage is to set all of the output pins LOW. This is the state you’d expect the system to be in when it first starts: everything is off. This is done using the OUTPUT and LOW commands. There is then a 200ms delay to make sure that everything has settled to the state that it’s been set to.

You should not have to change the initialization part of the code aside from tweaking the delay times.

Main Program Loop

The main program loop is the simplest part of the code. It calls the subroutines in the order required to complete the task. The names of the subroutines are pretty descriptive, so reading the code should make it clear what each is supposed to do. The main loop is an infinite loop, your crane will continue to operate until power is disconnected.

Subroutines

The subroutines are the meat of the code. Each one performs a specific action, which is then called at the right time to accomplish a complex task. The magnet control subroutines are given to you as an example. How they work will be described in the tutorial.
Appendix A – Skeleton Code

`{STAMP BS2}`
`{SPBASIC 2.5}`

Program: DesignDay.bs2
Version: 1.0
Description: This program is a starting point for controlling the Design Day crane.

Pin, Variable, and Constant Declarations

--- Pins used: ---
P7 = Electromagnet On
P5 = Motor Enable
P1 = Motor Control 1
P3 = Motor Control 2
P14 = pushButton Input – ACTIVE LOW

--- Constants: ---
magnet CON 7
motorEnable CON 5
motorInputOne CON 1
motorInputTwo CON 3

--- Variables: ---
pushButton VAR IN14

--- Delays: ---
GrabScrapDelay CON 500
LiftMagnetDelay CON 5000
LowerToTruckDelay CON 3000
ReleaseScrapDelay CON 500
TruckLeavesDelay CON 1000
WaitForTruck CON 1000
LowerToScrapDelay CON 1000
' -----------------------------------------------------------------------
' Initialization Code (set all outputs to a known state)
' -----------------------------------------------------------------------
INIT:
  OUTPUT magnet
  LOW magnet

  OUTPUT motorEnable
  LOW motorEnable

  OUTPUT motorInputOne
  LOW motorInputOne

  OUTPUT motorInputTwo
  LOW motorInputTwo

  PAUSE 200   ' wait 200 ms to system to stabilize

' -----------------------------------------------------------------------
' Main Program Loop
' -----------------------------------------------------------------------
MAIN:
  DEBUG "Starting The Main Loop",CR

  'Fill With Program Flow Subroutines

  PAUSE 1000 ' kill some time before looping again... adjust!
  GOTO MAIN ' keep going through main loop

' -----------------------------------------------------------------------
' Subroutines
' -----------------------------------------------------------------------
' Magnet Control Subroutines
' -----------------------------------------------------------------------
MagnetOn:
  HIGH magnet
  RETURN

MagnetOff:
  LOW magnet
  RETURN
'Motor Control Functions

MotorOn:
 'This Subroutine is Empty!
RETURN

MotorOff:
 'This Subroutine is Empty!
RETURN

MotorForward:
 'This Subroutine is Empty!
RETURN

MotorReverse:
 'This Subroutine is Empty!
RETURN

'Program Flow Functions

GrabScrap: 'Get the magnet to grab some scrap
  DEBUG "Grabbing Scrap",CR
 'This Subroutine is Empty!
RETURN

LiftMagnet: 'Set the magnet high and raise it
  DEBUG "Lifting Magnet",CR
 'This Subroutine is Empty!
RETURN

LowerToTruck: 'Lower the magnet to the truck
  DEBUG "Lowering to the Truck",CR
 'This Subroutine is Empty!
RETURN

ReleaseScrap: 'Release the scrap
  DEBUG "Releasing Scrap",CR
 'This Subroutine is Empty!
RETURN

TruckLeaves: 'Allow the truck to leave
  DEBUG "Truck Leaving",CR
 'This Subroutine is Empty!
RETURN
LowerToScrap: 'Return the magnet to the scrap
   DEBUG "Lowering to the Scrap",CR
   'This Subroutine is Empty!
RETURN